Operators

Operator is a symbol that performs certain operations.

Python provides the following set of operators

1. Arithmetic Operators
2. Relational Operators or Comparison Operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Special operators

1. Arithmetic Operators:

+ ==>Addition

- ==>Subtraction

\* ==>Multiplication

/ ==>Division operator

% ===>Modulo operator

// ==>Floor Division operator

\*\* ==>Exponent operator or power operator

Eg: test.py:

1

)

a=

10

2

)

b=

2

3

)

print

(

'a

+

b='

,a+b)

4

)

print

(

'a

-

b='

,a

-

b)

5

)

print

(

'a\*b='

,a\*b)

6

)

print

(

'a/b='

,a/b)

7

)

print

(

'a//b='

,a//b)

8

)

print

(

'a%b='

,a%b)

9

)

print

(

'a\*\*b='

,a\*\*b)

Output:

1

)

Python

test.py

or

py

test.py

2

)

a+b=

12

3

)

a

-

b=

8

4

)

a\*b=

20

5

)

a/b=

5.0

6

)

a//b=

5

7

)

a%b=

0

8

)

a\*\*b=

100

Eg:

1

)

a

=

10.5

2

)

b=

2

3

)

4

)

a+b=

12.5

5

)

a

-

b=

8.5

6

)

a\*b=

21.0

7

)

a/b=

5.25

8

)

a//b=

5.0

9

)

a%b=

0.5

10

)

a\*\*b=

110.25

Eg:

10/2==>5.0

10//2==>5

10.0/2===>5.0

10.0//2===>5.0

Note: / operator always performs floating point arithmetic. Hence it will always returns float value.

But Floor division (//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If at least one argument is float type then result is float type.

Note:

We can use +,\* operators for str type also.

If we want to use + operator for str type then compulsory both arguments should be str type only otherwise we will get error.

2)Type Error: must be str, not int

1

)

>>>

"jaga"

+

10

# 3)>>> "jaga"+"10"

# 4)'jag10'

If we use \* operator for str type then compulsory one argument should be int and other argument should be str type.

2\*"jaga"

"jaga"\*2

2.5\*"jaga" ==>Type Error: can't multiply sequence by non-int of type 'float'

"jaga"\*"jaga"==>Type Error: can't multiply sequence by non-int of type 'str'

+====>String concatenation operator

\* ===>String multiplication operator

Note: For any number x,

x/0 and x%0 always raises "ZeroDivisionError"

10/0

10.0/0

.....

Relational Operators:

>,>=,<,<=

Eg 1:

1

)

a=

10

2

)

b=

20

3

)

print

(

"a

>

b

is

"

,a>b)

4

)

print

(

"a

>=

b

is

"

,a>=b)

5

)

print

(

"a

<

b

is

"

,a<b)

6

)

print

(

"a

<=

b

is

"

,a<=b)

7

)

8

)

a

>

b

is

False

9

)

a

>=

b

is

False

10

)

a

<

b

is

True

11

)

a

<=

b

is

True

We can apply relational operators for str types also

Eg 2:

# 3) print("a > b is ",a>b)

1

)

a=

"jaga"

2

)

b=

"jaga"

# 4) print("a >= b is ",a>=b) 5) print("a < b is ",a<b)

6

)

print

(

"a

<=

b

is

"

,a<=b)

7

)

8

)

a

>

b

is

False

9

)

a

>=

b

is

True

10

)

a

<

b

is

False

11

)

a

<=

b

is

True

Eg:

1

)

print

(

True

>

True

)

False

2

)

print

(

True

>=

True

)

True

3

)

print

(

10

>

True

)

True

4

)

print

(

False

>

True

)

False

5

)

6

)

print

(

10

>

'jaga'

)

7

)

Type Error:

'>'

not

supported

between

instances

of

'int'

and

'str'

Eg:

1

)

a=

10

2

)

b=

20

3

)

if

(

a>b

):

4

)

print

(

"a

is

greater

than

b"

)

5

)

else

:

6

)

print

(

"a

is

not

greater

than

b"

)

Output a is not greater than b

Note: Chaining of relational operators is possible. In the chaining, if all comparisons returns True then only result is True. If at least one comparison returns False then the result is False

Eg:

1

)

10

<

20

==>

True

2

)

10

<

20

<

30

==>

True

3

)

10

<

20

<

30

<

40

==>

True

4

)

10

<

20

<

30

<

40

>

50

==>

False

equality operators:

== , !=

We can apply these operators for any type even for incompatible types also

1. >>> 10==20
2. False
3. >>> 10!= 20

|  |
| --- |
| 4) True |
| 5) >>> 10==True |
| 6) False |
| 7) >>> False==False |
| 8) True |
| 9) >>> "jaga"=="jaga" |
| 10) True |
| 11) >>> 10=="jaga " |
| 12) False |

Note: Chaining concept is applicable for equality operators. If at least one comparison returns False then the result is False. otherwise the result is True.

Eg:

1

)

>>>

10

==

20

==

30

==

40

2

)

False

3

)

>>>

10

==

10

==

10

==

10

4

)

True

Logical Operators:

and, or ,not

We can apply for all types.

For boolean types behaviour:

and ==>If both arguments are True then only result is True

or ====>If atleast one argument is True then result is True not ==>complement

True and False ==>False True or False ===>True

not False ==>True

For non-boolean types behaviour:

0 means False non-zero means True

empty string is always treated as False

x and y:

==>if x is evaluates to false return x otherwise return y

Eg:

10 and 20

0 and 20

If first argument is zero then result is zero otherwise result is y

x or y:

If x evaluates to True then result is x otherwise result is y

10 or 20 ==> 10

0 or 20 ==> 20

not x:

If x is evaluates to False then result is True otherwise False

not 10 ==>False

not 0 ==>True

Eg:

1

)

"jaga"

and

"durgasoft"

==>

durgasoft

2

)

"

"

and

"

durga

"

==>

"

"

3

)

"durga"

and

"

"

"

==>

"

4

)

"

"

or

"

durga

"

"

==>

durga"

5

)

"durga"

or

"

"==>"

durga"

6

)

not

""==>

True

7

)

not

"durga"

==>

False

Bitwise Operators:

We can apply these operators bitwise.

These operators are applicable only for int and Boolean types.

By mistake if we are trying to apply for any other type then we will get Error.

&,|,^,~,<<,>>

print(4&5) ==>valid print(10.5 & 5.6) ==>

Type Error: unsupported operand type(s) for &: 'float' and 'float'

print(True & True) ==>valid

& ==> If both bits are 1 then only result is 1 otherwise result is 0

| ==> If at least one bit is 1 then result is 1 otherwise result is 0

^ ==>If bits are different then only result is 1 otherwise result is 0

~ ==>bitwise complement operator

1==>0 & 0==>1

<< ==>Bitwise Left shift

>> ==>Bitwise Right Shift

print(4&5) ==>4 print(4|5) ==>5 print(4^5) ==>1

|  |  |
| --- | --- |
| Operator | Description |
| & | If both bits are 1 then only result is 1 otherwise result is 0 |
| | | If atleast one bit is 1 then result is 1 otherwise result is 0 |
| ^ | If bits are different then only result is 1 otherwise result is 0 |
| ~ | bitwise complement operator i.e 1 means 0 and 0 means 1 |
| >> | Bitwise Left shift Operator |
| << | Bitwise Right shift Operator |

bitwise complement operator(~):

We have to apply complement for total bits.

Eg: print(~5) ==>-6

Note:

The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represents -ve value. positive numbers will be represented directly in the memory where as -ve numbers will be represented indirectly in 2's complement form.

Shift Operators:

<< Left shift operator

After shifting the empty cells we have to fill with zero

print(10<<2)==>40

0

0

0

0

1

0

1

0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

>> Right Shift operator

After shifting the empty cells we have to fill with sign bit.( 0 for +ve and 1 for -ve)

print(10>>2) ==>2

0

0

0

0

1

0

1

0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

We can apply bitwise operators for boolean types also

print(True & False) ==>False print(True | False) ===>True print(True ^ False) ==>True

print(~True) ==>-2 print(True<<2) ==>4 print(True>>2) ==>0

Assignment Operators:

We can use assignment operator to assign value to the variable.

Eg:

x=10

We can combine assignment operator with some other operator to form compound assignment operator.

Eg: x+=10 ====> x = x+10

The following is the list of all possible compound assignment operators in Python

+=

-=

\*=

/=

%=

//=

\*\*=

&=

|=

^=

>>=

<<=

Eg:

1

)

x=

10

2

)

x+=

20

3

)

print

(

x

)

==>

30

E.g.:

1

)

x=

10

2

)

x&=

5

3

)

print

(

x

)

==>

0

Ternary Operator:

Syntax:

x = first Value if condition else second Value

If condition is True then first Value will be considered else second Value will be considered.

Eg 1:

1

)

a,b=

10

,

20

2

)

x=

30

if

a<b

else

40

3

)

print

(

x

)

#30

Eg 2: Read two numbers from the keyboard and print minimum value

1) a=int(input("Enter First Number:")) 2) b=int(input("Enter Second Number:"))

3) min=a if a<b else b

# 4) print("Minimum Value:",min)

Output:

Enter First Number:10

Enter Second Number:30

Minimum Value: 10

Note: Nesting of ternary operator is possible.

Q. Program for minimum of 3 numbers

# 1) a=int(input("Enter First Number:")) 2) b=int(input("Enter Second Number:")) 3) c=int(input("Enter Third Number:"))

1. min=a if a<b and a<c else b if b<c else c
2. print("Minimum Value:”, in)

Q. Program for maximum of 3 numbers

# 1) a=int(input("Enter First Number:")) 2) b=int(input("Enter Second Number:")) 3) c=int(input("Enter Third Number:"))

1. max=a if a>b and a>c else b if b>c else c
2. print("Maximum Value:",max)

Eg:

# 1) a=int(input("Enter First Number:")) 2) b=int(input("Enter Second Number:")) 3) print("Both numbers are equal" if a==b else "First Number is Less than Second Number" if a<b else "First Number Greater than Second Number")

Output:

D:\python\_classes>py test.py

Enter First Number:10

Enter Second Number:10

Both numbers are equal

D:\python\_classes>py test.py

Enter First Number:10

Enter Second Number:20

First Number is Less than Second Number

D:\python\_classes>py test.py

Enter First Number:20

Enter Second Number:10

First Number Greater than Second Number

Special operators:

Python defines the following 2 special operators

1. Identity Operators
2. Membership operators

1. Identity Operators

We can use identity operators for address comparison.

2 identity operators are available

1. is
2. is not

r1 is r2 returns True if both r1 and r2 are pointing to the same object r1 is not r2 returns True if both r1 and r2 are not pointing to the same object

Eg:

1

)

a=

10

2

)

b=

10

3

)

print

(

a

is

b)

T

rue

4

)

x=

True

5

)

y=

True

6

)

print

(

x

is

y)

True

Eg:

1

)

a=

"durga"

2

)

b=

"durga"

3

)

print

(

id(a

))

4

)

print

(

id(b

))

5

)

print

(

a

is

b)

Eg:

1

)

list1=[

"one"

,

"two"

,

"three"

]

2

)

list2=[

"one"

,

"two"

,

"three"

]

3

)

print

1))

(

id(list

4

)

print

2))

(

id(list

5

)

print

(

list

1

is

list2)

Fals

e

6

)

print

(

list

1

is

not

list2)

True

7

)

print

(

list

1

==

list2)

True

Note:

We can use is operator for address comparison where as == operator for content comparison.

2. Membership operators:

We can use Membership operators to check whether the given object present in the given collection.(It may be String,List,Set,Tuple or Dict)

in  Returns True if the given object present in the specified Collection not in  Retruns True if the given object not present in the specified Collection

Eg:

1

)

x=

"hel

lo

learning

Python

is

very

easy!!!"

2

)

print

(

'h'

in

x)

True

3

)

print

(

'd'

in

x)

False

4

)

print

(

'd'

not

in

x)

True

5

)

print

(

'

Python

'

in

x)

True

Eg:

# 1) list1=["sunny","bunny","chinny","pinny"]

1. print("sunny" in list1) True
2. print("tunny" in list1) False
3. print("tunny" not in list1) True

Operator Precedence:

If multiple operators present then which operator will be evaluated first is decided by operator precedence.

Eg:

print(3+10\*2)  23

print((3+10)\*2)  26

The following list describes operator precedence in Python

()  Parenthesis

\*\*  exponential operator

~,-  Bitwise complement operator, unary minus operator

\*,/,%,//  multiplication,division,modulo,floor division

+,-  addition,subtraction

<<,>>  Left and Right Shift

&  bitwise And

^  Bitwise X-OR

|  Bitwise OR

>,>=,<,<=, ==, != ==>Relational or Comparison operators

=,+=,-=,\*=... ==>Assignment operators is , is not  Identity Operators in , not in  Membership operators not  Logical not and  Logical and

or  Logical or

Eg:

1

)

a=

30

2

)

b=

20

3

)

c=

10

4

)

d=

5

5

)

print

((

a+b)\*c/d

)

100.0

6

)

print

((

a+b)\*(c/d

))

100.0

7

)

print

(

)

a+(b\*c)/d

70.0

8

)

9

)

10

)

3

/

2

\*

4

+

3

+(

10

/

5

)\*\*

3

-

2

11

)

3

/

2

\*

4

+

3

+

2.0

\*\*

3

-

2

12

)

3

/

2

\*

4

+

3

+

8.0

-

2

13

)

1.5

\*

4

+

3

+

8.0

-

2

14

)

6.0

+

3

+

8.0

-

2

15

)

15.0

Mathematical Functions (math Module)

A Module is collection of functions, variables and classes etc.

math is a module that contains several functions to perform mathematical operations

If we want to use any module in Python, first we have to import that module.

import math

Once we import a module then we can call any function of that module.

import math print(math.sqrt(16))

print(math.pi)

4.0

3.141592653589793

We can create alias name by using as keyword.

import math as m

Once we create alias name, by using that we can access functions and variables of that module

import math as m print(m.sqrt(16))

print(m.pi)

We can import a particular member of a module explicitly as follows

from math import sqrt

from math import sqrt,pi

If we import a member explicitly then it is not required to use module name while accessing.

from math import sqrt,pi

print(sqrt(16)) print(pi)

print(Mathia)  Name Error: name 'math' is not defined

important functions of math module:

ceil(x) floor(x) pow(XXY) factorial(x) trunk(x) gcd(XXY) sin(x) cos(x) tan(x)

....

important variables of math module:

pi3.14 e===>2.71

inf ==>infinity

nan ==>not a number

Q. Write a Python program to find area of circle

pi\*r\*\*2

from math import pi r=16

print("Area of Circle is :",pi\*r\*\*2)

OutputArea of Circle is : 804.247719318987